Minimax Theory and its Applications
Volume 11 (2026), No. 3s, 467 to 482

Residual Learning-Based Approximate Deficit Round Robin Scheduling for Fair and Reconfiguration-Aware Task Management in Multi-Tenant FPGA Systems

Abirami B[1*], V. Vasudevan[2]

[1,2]Department of Computer Science and Engineering, Kalasalingam Academy of Research and Education, Krishnankoil, Tamil Nadu, India
Email: nbs.abirami@gmail.com , drvasudeva.srivi@gmail.com
*Corresponding author

Abstract
Multi-tenant field-programmable gate arrays (FPGAs) share a few partially reconfigurable regions among tenants whose hardware tasks run non-preemptively, whose execution times are known only through inaccurate user estimates, and whose every change of resident bitstream costs a serialised reconfiguration. This paper presents RL-A-DRR, a residual-learning approximate deficit round robin scheduler. A 177-parameter neural predictor learns the log residual between the requested and the actual execution time from causal user-history features and is refined online. Its corrected costs, augmented with the expected reconfiguration delay, drive a deficit round robin over four length–priority classes whose admission is approximate: only a look-ahead window of W tasks per class is examined, so a decision is independent of queue length, and credit-eligible tasks whose bitstream is already resident are preferred and may borrow a bounded credit. Deficits are trued-up with the actual occupancy, which bounds the weighted service gap irrespective of prediction bias, while congestion-adaptive aging and a starvation guard limit bypassing. RL-A-DRR is evaluated with an open C simulator driven by four production traces of the Parallel Workloads Archive (403,005 jobs) mapped onto a four-region FPGA with a 400 MB/s configuration port. Over 40 windows of 5,000 tasks at 80% load, RL-A-DRR reduces the mean waiting time by 70.8% relative to first-come first-served (FCFS) and by 46.5–64.6% relative to reconfiguration-aware FCFS, classic deficit round robin and multilevel feedback queueing; it matches shortest-job-first on mean waiting time while lowering its 99th-percentile waiting time by 50–62%, and it cuts reconfigurations by 39% and the premium tier's mean waiting time by 87.5% relative to FCFS. Under a tenfold under-estimation of low-priority costs, error compensation keeps the premium-tier waiting time within 38% of its unbiased value, whereas uncompensated deficits inflate it 1.8–4.6 times. A scheduling decision takes 0.80 μs on average.

Keywords: FPGA scheduling; partial reconfiguration; deficit round robin; residual learning; runtime prediction; starvation mitigation; multi-tenancy.

Introduction

Partially reconfigurable FPGAs are now shared by several tenants in data centres and at the edge. Production shells divide the fabric into a static region and a few reconfigurable partitions that are loaded with tenant accelerators on demand through dynamic function exchange [1, 2]. Recent systems raise the abstraction level of this sharing: Coyote v2 provides multithreaded and multi-tenant services on data-centre FPGAs [3], Funky orchestrates FPGA workloads in cloud-native sandboxes with preemption and migration [4], VersaSlot combines heterogeneous Big.Little slots with live migration [5], and THEMIS schedules tenants for time-, heterogeneity- and energy-aware fairness [6]. Scheduling, however, remains constrained by three properties that general-purpose schedulers do not model. First, hardware tasks are effectively non-preemptive: context saving on FPGAs is possible but costly and vendor-specific [7, 8]. Second, every change of the resident bitstream costs a reconfiguration whose duration scales with the partial bitstream size and which is serialised by a single configuration port [9]. Third, the scheduler sees only an estimate of each task's execution time; in production logs these estimates are poor, and users typically over-request by large factors [10, 11].

These properties make fairness and efficiency hard to reconcile. Shortest-job-first (SJF) ordering minimises mean waiting time but starves long tasks, and it inherits every error of the estimates it sorts on. Fair queueing schedulers such as deficit round robin (DRR) bound the service gap between classes [12], but the bound is stated

in units of charged cost; when the cost is a biased estimate, the scheduler is fair with respect to the wrong quantity. Reconfiguration-aware schedulers reduce bitstream reloads by reusing resident kernels [13, 14], yet greedy reuse ignores priorities and fairness. Learning-augmented schedulers predict job sizes [15–17], but they rarely couple predictions to a fair-share mechanism whose guarantee survives prediction bias, and none of them accounts for reconfiguration.

This article develops residual-learning approximate DRR (RL-A-DRR), a scheduler that addresses these three properties together for multi-tenant partially reconfigurable FPGAs. Its contributions are as follows.

- **Residual cost model.** A 9–16–1 neural network predicts the log residual $\ln(e/\tilde{b})$ between the actual execution time e and the user request $\tilde{b}$ from strictly causal features, is refined online after each completion, and is combined with the expected reconfiguration delay into a corrected admission cost (Section 4.1).
- **Approximate, reconfiguration-affine DRR admission.** Admission inspects a window of W tasks per class, prefers credit-eligible tasks whose kernel is resident, and lets them borrow at most B quanta, so a decision costs $O(CWS)$ for C classes and S partitions, independent of the queue length (Section 4.2).
- **Error-compensated deficits.** Charging the predicted cost and truing up with the measured occupancy yields a service-gap bound that does not grow with cumulative prediction error (Proposition 1); we show experimentally that this protects premium tenants under biased predictions.
- **Congestion-adaptive anti-starvation.** Aging credit and a starvation guard use thresholds relative to the running mean waiting time, which avoids the collapse to FCFS that absolute thresholds cause under overload (Section 4.4).
- **Trace-driven open evaluation.** A C simulator, four production traces with 403,005 jobs, six baselines, statistical testing with multiplicity correction, ablations, sensitivity, scalability, overhead and heterogeneous-partition studies (Sections 5–6).

Section 2 reviews related work, Section 3 defines the system model, Section 4 presents RL-A-DRR, Sections 5 and 6 describe the methodology and results, Section 7 discusses validity and limitations, and Section 8 concludes.

Related work

Scheduling on shared partially reconfigurable FPGAs

Operating-system support for shared fabric has matured from AmorphOS [18] and cloud FPGA virtualisation [19] to Coyote [20] and Coyote v2 [3]. Toolflows such as TaPaSCo compose task-parallel accelerators [21], and lightweight runtimes co-schedule CPU and FPGA work [22]. DML partitions applications and schedules them with dynamic partial reconfiguration [14]; Nimblock virtualises slots and shares them at fine granularity [13]; VersaSlot shows that reconfiguration contention dominates response time and mitigates it with Big.Little slots and asynchronous reconfiguration [5]. THEMIS assigns slots per scheduling interval according to tenants' area–time demand and reconfigures only when the tenant changes, exposing a knob between energy and fairness [6]. Jiang et al. move adaptive spatio-temporal scheduling into hardware and report a $>1000\times$ faster decision than software [23]. Preemption is being made practical through context saving [7], unikernel checkpointing [4] and unified accelerator command queues [8], but these mechanisms add state-transfer latency; a recent survey of real-time scheduling on accelerator platforms notes that most FPGA schedulers still assume known execution times [24]. Reinforcement learning has been applied to FPGA configuration selection [25] and to fair firm-real-time scheduling of multi-tenant accelerators [26], at the cost of training effort and opaque guarantees.

Fair and credit-based scheduling

DRR achieves $O(1)$ fair queueing with a per-flow deficit counter [12]. Self-clocked round robin revisits DRR for short latency-sensitive flows [27], and the virtual token counter brings work-conserving, provably fair scheduling to large-language-model serving by charging tokens as they are processed [28]. The last work is closest in spirit to our true-up: service is measured, not assumed. Jain's index [29] remains the common fairness summary.

Learning-augmented scheduling and runtime prediction

Algorithms with predictions seek consistency when predictions are good and robustness when they are not [15]; recent results cover non-clairvoyant scheduling with partial predictions [30] and competitive fair scheduling under prediction error [31]. In systems, learning-to-rank approximates SJF for LLM requests [16], prediction-

assisted scheduling serves GPU clusters [17, 32], and reinforcement learning schedules and backfills HPC batch jobs [33, 34]. Runtime prediction from user history dates back to Tsafrir et al. [10]; tandem models [35] and gradient-boosted models on 7.3 million NREL jobs [11] show that accuracy drops sharply under chronological evaluation, which motivates our online refinement. For FPGA tasks, graph neural networks predict post-route quality of results from HLS sources [36], which could seed the request $\tilde{b}$ used here. Table 1 positions RL-A-DRR against representative work.

Table 1. Positioning of RL-A-DRR (✓ supported, ○ partial, – absent).

Work	Non-preempt.	Reconf.-aware	Fair share	Learned cost	Starv. bound	Bias-robust
DML [14], Nimblock [13]	○	✓	–	–	–	–
VersaSlot [5]	○	✓	–	–	–	–
THEMIS [6]	○	✓	✓	–	○	–
Russo et al. [26]	○	–	✓	○ (RL)	–	–
DRR [12], SCRR [27]	✓	–	✓	–	✓	–
VTC [28]	○	–	✓	–	✓	✓
LTR [16], Zhao et al. [31]	○	–	○	✓	○	○
RL-A-DRR (this work)	✓	✓	✓	✓	✓	✓

System model and problem statement

Platform. The FPGA exposes S partially reconfigurable regions (PRRs) $\{1, \dots, S\}$ behind a static shell. Region s has a relative compute rate v_s, a partial bitstream of M_s megabytes and a dynamic-power factor φ_s. Bitstreams are written through a single configuration port (ICAP) of bandwidth β, so a reconfiguration of s lasts $T_s = M_s/\beta$ and reconfigurations are serialised. A region holds one of K accelerator kernels or is blank.

Tasks. Task j arrives at a_j from tenant u_j with service tier $\pi_j \in \{\mathrm{H}, \mathrm{L}\}$, needs kernel k_j, carries a user request $\tilde{b}_j$ and has an unknown actual execution time e_j at unit rate. If j is dispatched to region s at time d_j, it first waits for the ICAP and reconfigures if the region does not hold k_j, then executes non-preemptively for e_j/v_s; its execution starts at x_j and finishes at f_j. The region occupancy is $o_j = f_j - d_j$.

Metrics. The waiting time is $w_j = x_j - a_j$ (it includes reconfiguration delays), the turnaround time is $f_j - a_j$, and the bounded slowdown [37] is $\max\{1, (f_j - a_j)/\max(f_j - x_j, \tau_b)\}$ with $\tau_b = 1$ ms. We report mean, 99th-percentile and maximum waiting times, the reconfiguration ratio, throughput, energy per task, Jain's index [29] over the mean bounded slowdown of four ground-truth categories (execution time below/above the window median × tier), and the number of starved tasks, i.e., tasks with $w_j > \theta_s$. Energy is

$$E = P_{\mathrm{st}} \left(\max_j f_j - \min_j a_j \right) + \sum_j P_{k_j} \varphi_{s_j} \frac{e_j}{v_{s_j}} + P_{\mathrm{rc}} \sum_{\text{reconf.}} T_s \,, \qquad (1)$$

with static, kernel and reconfiguration powers P_{st}, P_k and P_{rc}.

Problem. Online and without knowledge of e_j, choose at each region release a waiting task and a free region so as to minimise waiting and slowdown and the number of reconfigurations, subject to (i) weighted fairness between the four length–priority classes SJHP, SJLP, LJHP and LJLP (short/long × high/low priority) with weights w_c, and (ii) no indefinite postponement of any task.

The RL-A-DRR scheduler

Fig. 1 shows the pipeline. A task's request and causal features are mapped by the residual predictor to a corrected execution estimate; the classifier places the task in one of four class queues; the admission engine selects a task from look-ahead windows using deficit credit, kernel residency, aging and the starvation guard; the partition selector chooses a region; and completions feed back the measured occupancy and residual.

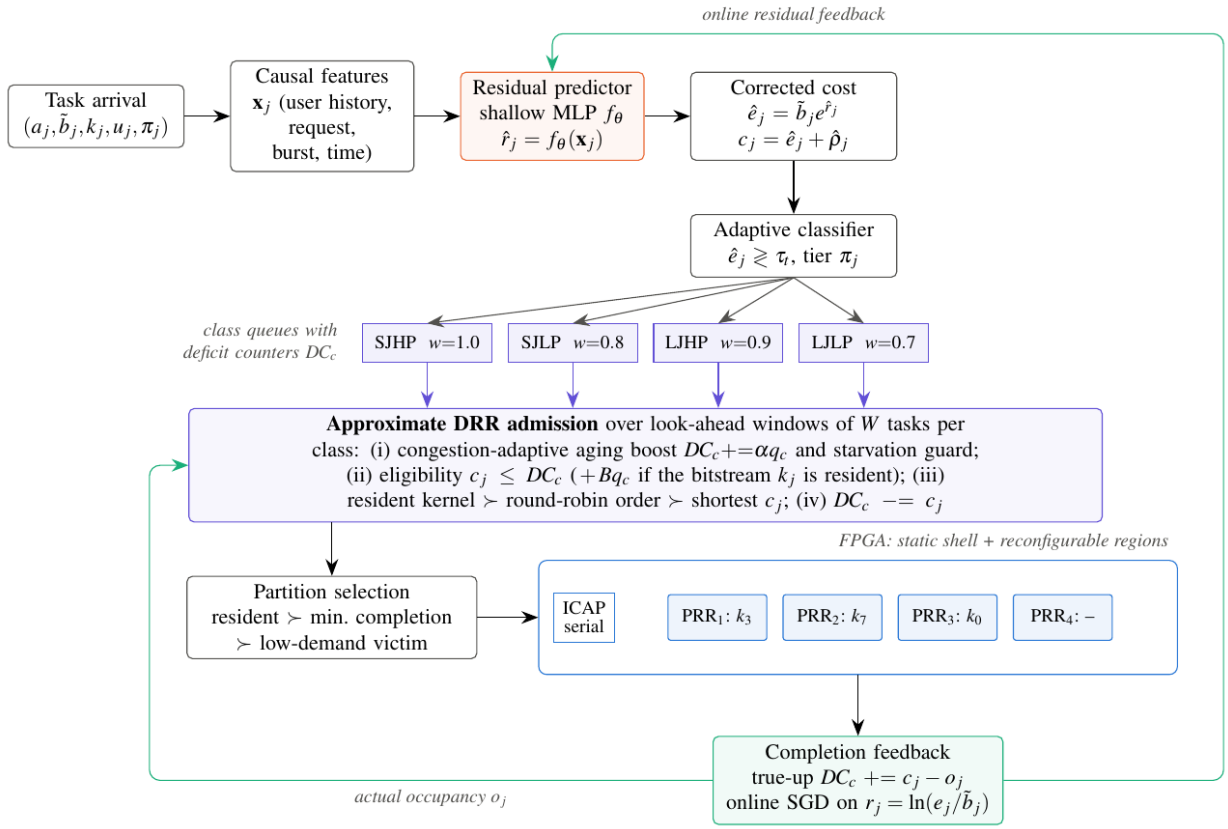


Fig. 1. Architecture of RL-A-DRR. Orange: learning; violet: approximate DRR admission; blue: FPGA; green: completion feedback (error compensation and online learning).

Residual predictor and corrected cost

Rather than predicting e_j directly, the predictor estimates the multiplicative residual of the request, $r_j = \ln(e_j/\tilde{b}_j)$, clipped to $[-10,2]$. The log form makes the target scale-free across tasks whose lengths span six orders of magnitude, and residual learning lets the network start from the user's information instead of replacing it. The input $\mathbf{x}_j \in \mathbb{R}^9$ is computed at arrival from information that is available at that instant: $\ln\tilde{b}_j$, $\log_2$ of the requested parallelism, the log execution times of the tenant's two most recent *completed* tasks, an exponentially weighted mean of the tenant's past residuals, the log of a decayed count of the tenant's recent submissions (time constant 1 h), the sine and cosine of the submission hour, and a history flag. The model f_θ is a multilayer perceptron with 9 inputs, 16 ReLU units and a linear output (177 parameters), trained with mean squared error and Adam [38] (learning rate 10^{-3}, batch 32, early stopping with patience 6). After each completion, one stochastic-gradient step (rate 2×10^{-4}, gradient clipping at 1) on the observed residual adapts the model to drift. The corrected execution estimate and the admission cost are

$$\hat{e}_j = \tilde{b}_j\, e^{\hat{r}_j}, \qquad c_j = \hat{e}_j + \hat{\rho}_j, \qquad \hat{\rho}_j = \begin{cases} 0, & k_j \text{ resident in a free region,} \\ \min_{s \text{ free}} \left(\max\left(0, t_{\text{icap}} - t\right) + T_s\right), & \text{otherwise,} \end{cases} \tag{2}$$

where t_{icap} is the time at which the ICAP becomes free. Charging $\hat{\rho}_j$ makes tenants pay for the reconfigurations they cause.

Adaptive classification. A task is *long* if $\hat{e}_j \ge \tau_t$, where $\ln\tau_t$ is an exponentially weighted moving average (factor 0.01) of $\ln\hat{e}$ over arrivals, i.e., a running geometric mean. Unlike a threshold based on the mean burst time of all tasks, this is causal. The class is (short/long) $\times\, \pi_j$ with weights $w = (1.0,0.8,0.9,0.7)$ for (SJHP, SJLP, LJHP, LJLP). The quantum of class c is $q_c = w_c\, Q$, where Q is a moving average of $\hat{e}$, so quanta follow the workload scale.

Approximate DRR admission with reconfiguration affinity

Each class keeps a FIFO queue and a deficit counter DC_c. As in DRR, a round-robin pointer visits the classes and grants q_c to a backlogged class once per visit; the pointer moves only when no task is eligible. Two departures from DRR make the admission *approximate* and *reconfiguration-aware*. First, only the first W tasks of each queue are inspected, which bounds the work per decision independently of queue length. Second, eligibility is evaluated over all windows: task j of class c is eligible if $c_j \le DC_c$, or if k_j is resident in a free region and $c_j \le DC_c + B\,q_c$ (bounded *affinity borrowing*). Among eligible tasks, the rule prefers resident kernels, then the class nearest to the pointer, then the smallest c_j inside that class's window. Algorithm 1 gives the decision procedure. When a whole round finds nothing eligible, the number of rounds $k^* = \min_c \lceil (\min_{j \in \mathcal{W}_c} c_j - DC_c)/q_c \rceil$ needed by the first class is computed and $(k^* - 1)q_c$ is granted to every backlogged class at once, which is equivalent to running the empty rounds.

Partition selection. A task whose kernel is resident runs in that region. Otherwise RL-A-DRR chooses the free region with the smallest expected completion $\max(0, t_{\text{icap}} - t) + T_s + \hat{e}_j / v_s$; ties are broken by evicting the kernel with the lowest pending demand in the windows, and then by least-recent use.

Algorithm 1 RL-A-DRR decision at a region release (time t)

Require: class queues $\mathcal{Q}_c$ with windows $\mathcal{W}_c$ (first W tasks), deficits DC_c, pointer p, mean wait $\bar{w}$

$\theta_a \leftarrow \max(\theta_a^0, \kappa_a \bar{w})$; $\theta_g \leftarrow \max(\theta_g^0, \kappa_g \bar{w})$ ▷ congestion-adaptive thresholds

for each class c and each $j \in \mathcal{W}_c$ **do**

  while $t \ge \text{next_boost}_j$ **do** $DC_c \leftarrow DC_c + \alpha q_c$; $\text{next_boost}_j \leftarrow \text{next_boost}_j + \theta_a$

if some $j \in \bigcup_c \mathcal{W}_c$ has $t - a_j > \theta_g$ **then**

  $j^* \leftarrow$ oldest such task; **goto** 12 ▷ starvation guard

loop

  if $\mathcal{Q}_p = \emptyset$ **then** $DC_p \leftarrow \min(DC_p, 0)$; $p \leftarrow p + 1$; **continue**

  if class p not yet credited in this visit **then** $DC_p \leftarrow DC_p + q_p$

  $\mathcal{E} \leftarrow \{ j \in \mathcal{W}_c : c_j \le DC_c + Bq_c \cdot [\hat{\rho}_j = 0] \}$ over classes $c = p, p+1, \ldots$ ▷ Eq. (2)

  if $\mathcal{E} \neq \emptyset$ **then** $j^* \leftarrow$ first of $\mathcal{E}$ by (resident, class order, c_j); **goto** 12

  $p \leftarrow p + 1$; after a full unproductive round, fast-forward $(k^* - 1)$ rounds

remove j^* from its class c^*; $DC_{c^*} \leftarrow DC_{c^*} - c_{j^*}$; choose region; dispatch

at completion of j**:** $DC_c \leftarrow DC_c + c_j - o_j$; one SGD step on $(\mathbf{x}_j, r_j)$ ▷ error compensation

Error-compensated deficits

A task is admitted with its predicted cost c_j, but when it completes the counter is corrected by $c_j - o_j$, so each class is ultimately debited with the region time it actually consumed. When a queue empties, positive credit is discarded as in DRR, whereas debt is kept. Let $S_c(t_1, t_2)$ denote the occupancy of the class-c tasks dispatched in $[t_1, t_2]$.

Proposition 1. *Let classes i and j be continuously backlogged in $[t_1, t_2]$, let Q be constant, and suppose no aging credit or guard dispatch occurs. Let L be the largest admission cost and $U \ge 0$ the largest under-estimate $o - c$. With error compensation,*

$$\left| \frac{S_i(t_1,t_2)}{w_i} - \frac{S_j(t_1,t_2)}{w_j} \right| \le (3 + 2B)\,Q + (S+1)(L+U)\left(\frac{1}{w_i} + \frac{1}{w_j}\right). \qquad (3)$$

Without error compensation, the right-hand side gains the term $\left| \mathcal{E}_i/w_i - \mathcal{E}_j/w_j \right|$ with $\mathcal{E}_c = \sum_{\ell \in c} (o_\ell - c_\ell)$, which grows linearly with the number of dispatched tasks whenever predictions are biased.

Proof sketch. Quanta are granted only at pointer visits, and the pointer cycles through the classes, so the numbers of quanta n_i and n_j granted in the interval differ by at most one. With compensation, the counter that includes the pending corrections of in-flight tasks evolves as $DC'_c(t_2) = DC'_c(t_1) + n_c q_c - S_c$. A quantum is granted only when no window task is eligible, hence when $DC_c < L$, so $DC_c < L + q_c$ after a grant; refunds of at most S in-flight tasks add at most SL. A dispatch requires $c \le DC_c + Bq_c$, so $DC_c \ge -Bq_c$ afterwards, and the corrections of at most S in-flight tasks subtract at most SU. Hence DC'_c ranges over an interval of width at most $q_c(1 + B) + (S+1)(L+U)$. Dividing $S_c = n_c q_c - \Delta DC'_c$ by w_c, using $q_c = w_c Q$ and $|n_i - n_j| \le 1$ gives the bound. Without compensation, $S_c = n_c q_c - \Delta DC_c + \mathcal{E}_c$. □

The bound holds whatever the predictor's bias; prediction quality only affects *which* tasks are admitted first. Aging credit and guard dispatches deliberately deviate from strict shares; their effect is bounded by the boosts granted, and Section 6.4 quantifies it.

Congestion-adaptive aging and starvation guard

A window task that has waited θ_a earns its class a credit αq_c, and again after every further θ_a. A task waiting longer than θ_g bypasses the credit test and is dispatched at the next release, oldest first. Both thresholds are relative to the moving average $\bar{w}$ of observed waiting times (factor 0.01), $\theta_a = \max(10\bar{e}, \kappa_a \bar{w})$ and $\theta_g = \max(60\bar{e}, \kappa_g \bar{w})$, where $\bar{e}$ is the mean execution time. Fixed thresholds misbehave under overload: when most tasks exceed an absolute guard, the guard turns the scheduler into FCFS, which forfeits kernel reuse and *increases* the number of starved tasks; on the tuning trace an absolute guard of 2.5 s raised the starved tasks from 3 to 160 per window.

Remark 1 (No indefinite postponement). *The oldest task of every class is always inside its window, and the guard serves guard-eligible tasks in arrival order. Therefore a task can be bypassed only while its wait is below θ_g or while older guard-eligible tasks exist; in a stable system both phases are finite.*

Complexity. Inference costs about 160 multiply–accumulates. A decision scans at most CW tasks and evaluates the residency of each kernel over S regions, i.e., $O(CWS)$ with $C = 4$, independent of the number n of waiting tasks; the true-up is $O(1)$. The memory is one queue node per task, four counters and 177 parameters.

Experimental methodology

Workloads. We use four production logs of the Parallel Workloads Archive [39] in Standard Workload Format (Table 2). Each job record provides the submission time, the user, the requested parallelism, the user's requested time $\tilde{b}$ and the actual run time e; jobs with non-positive run or requested time are discarded. The logs differ markedly: the median ratio of actual to requested time ranges from 0.16 to 0.61, 7–18% of jobs exceed their request, and HPC2N contains large single-user submission bursts. A fifth log (SDSC-BLUE) is used *only* to tune the scheduler parameters and never for evaluation.

Table 2. Production traces. $\tilde{\rho}$: median $e/\tilde{b}$; Under: jobs with $e > \tilde{b}$; r: Pearson correlation of $\ln e$ and $\ln \tilde{b}$.

Trace	System	Period	Jobs	Users	Mean e (s)	$\tilde{\rho}$	Under (%)	r
SDSC-SP2	IBM SP2 (128 CPUs)	1998–2000	53,999	428	7,395	0.16	7.7	0.68
CTC-SP2	IBM SP2 (338 CPUs)	1996–1997	77,199	679	11,280	0.27	9.3	0.70
HPC2N	Linux cluster (240 CPUs)	2002–2006	202,871	257	17,024	0.47	7.3	0.52
ANL-Intrepid	Blue Gene/P (163,840 cores)	2009	68,936	236	5,176	0.61	17.8	0.67

Mapping to FPGA tasks. Each job becomes one hardware task. Its tenant is the user; the kernel is a fixed hash of (user, $\lfloor \log_2$ parallelism$\rfloor$) onto $K = 8$ bitstreams, which preserves the reuse locality of real submission streams; 30% of the users, selected by a fixed hash, form the premium tier. Times are scaled so that the mean execution time of the training half equals $\bar{e} = 25$ ms, and the arrival process of each window is compressed uniformly so that the offered load, $\sum_j e_j$ divided by the window span and the total capacity $\sum_s v_s$, equals ρ; burstiness and diurnal structure are preserved. The default platform has $S = 4$ homogeneous regions with 1.6 MB bitstreams and a 32-bit, 100 MHz ICAP ($\beta = 400$ MB/s), so $T_s = 4$ ms [1, 9]. Powers are $P_{\mathrm{st}} = 12$ W, $P_k = 2.0 + 0.4k$ W and $P_{\mathrm{rc}} = 1.5$ W; absolute energies are therefore indicative, and we compare policies relative to each other. The Big.Little variant has two regions with $v = 1.5$, $M = 3.2$ MB, $\varphi = 1.6$ and two with $v = 0.7$, $M = 0.8$ MB, $\varphi = 0.6$.

Protocol. The predictors are trained on the chronologically first half of each trace (85% training, 15% validation) and evaluated on the second half, as recommended for runtime prediction [11]. Every simulation window consists of $N = 5{,}000$ consecutive jobs from the test half, starting at a seed-dependent offset; ten seeds {42,56,78,90,102,115,127,133,149,161} give 40 windows (200,000 tasks) per configuration. Per-user histories are warmed causally up to the window start and then updated only by simulated completions. All policies see identical task sets. The default load is $\rho = 0.8$, the starvation threshold is $\theta_s = 200\,\bar{e} = 5$ s, and the

RL-A-DRR parameters, selected on SDSC-BLUE at loads 0.7–0.9, are $W = 16$, $B = 4$, $\kappa_a = 2$, $\kappa_g = 8$ and $\alpha = 1.25$. We compare paired windows with the Wilcoxon signed-rank test [40] and Holm's correction over all 24 policy–metric tests [41], and summarise relative changes by geometric means of per-window ratios.

Baselines. FCFS; SJF-E and SJF-P, i.e., non-preemptive shortest-job-first on the user request and on the same learned prediction as RL-A-DRR; RA-FCFS, a reconfiguration-aware FCFS that dispatches the first of the 16 oldest tasks whose kernel is resident, in the spirit of reuse-oriented FPGA schedulers [13, 14]; DRR [12] over the same four classes and weights with the request as cost; and MLFQ-A, a non-preemptive four-level feedback queue with aging promotion after $10\bar{e}$, representing MLFQ variants. All baselines reuse a resident kernel when the chosen task's kernel is loaded in a free region and otherwise evict the least recently used region.

Implementation. The simulator, the predictors and all policies are about 2,000 lines of C11 (gcc -O2); runs execute on an Intel Xeon processor at 2.8 GHz. Decision latencies are measured with the monotonic clock around every decision. The code, the trace checksums and the scripts that regenerate every table and figure are released with this article.

Results and discussion

Accuracy and cost of the residual predictor

Table 3 reports the errors of the log execution time on the 27,000–101,436 held-out jobs of each trace. User requests are biased by -1.06 to -2.10 log units, i.e., users over-request by 2.9–8.2 × in geometric mean, and explain less variance than a constant ($R^2 < 0$). Ridge regression and both networks remove this bias ($|\text{bias}| \leq 0.04$), and the Tsafrir heuristic reduces it to -0.19 or less in magnitude. The shallow MLP attains the lowest RMSE on all four traces and the lowest MAE on HPC2N and ANL-Intrepid, whereas the Tsafrir heuristic [10] has a lower MAE on SDSC-SP2 and CTC-SP2 because many users resubmit identical jobs. The deep network is never more accurate, although it has 42 times more parameters and is 40 times slower (7.5 μs against 0.19 μs per prediction). The modest R^2 (0.11–0.34) agrees with the chronological results reported on other HPC logs [11]; Fig. 2 shows that the residual model nevertheless centres the error distribution and narrows it relative to the request. RL-A-DRR does not need precise predictions, because the true-up removes their effect on long-run shares.

Table 3. Predictor accuracy on the chronological test halves (natural-log units) and inference latency. Shallow MLP: 9–16–1; deep MLP: 9–64–64–32–16–1.

	SDSC-SP2			**CTC-SP2**			**HPC2N**			**ANL-Intrepid**				**Lat.**
Predictor	MAE	RMSE	R^2	MAE	RMSE	R^2	MAE	RMSE	R^2	MAE	RMSE	R^2	Params	(ns)
User request	2.02	2.64	−1.40	1.75	2.51	−0.94	2.12	3.48	−0.57	1.08	1.62	−0.76	–	3
Tsafrir (last-2)	1.05	1.76	−0.06	1.15	1.98	−0.21	1.75	3.00	−0.18	0.65	1.20	0.03	–	29
Ridge regression	1.09	1.51	0.21	1.18	1.67	0.14	1.70	2.65	0.08	0.63	0.99	0.34	10	14
Shallow MLP (prop.)	1.10	1.51	0.22	1.16	1.66	0.15	1.66	2.61	0.11	0.63	0.99	0.34	177	188
Deep MLP	1.07	1.52	0.21	1.23	1.71	0.09	1.69	2.63	0.10	0.65	1.01	0.32	7425	7,518

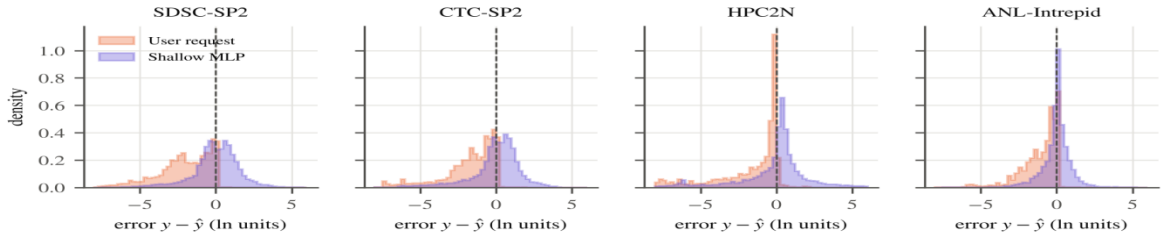


Fig. 2. Distribution of the log-time error of the user request and of the shallow residual predictor on the test halves.

Comparison with the baselines

Table 4 lists per-trace means over ten windows at $\rho = 0.8$, Fig. 3 summarises them relative to FCFS, and Table 5 gives the paired tests. RL-A-DRR has the lowest mean waiting time on SDSC-SP2 and CTC-SP2, is within 21% of SJF-P on ANL-Intrepid, and gives the premium tier the lowest waiting time on all four traces. Across the 40 windows it reduces the mean waiting time by 70.8% relative to FCFS, 64.6% relative to MLFQ-A, 54.7% relative to DRR and 46.5% relative to RA-FCFS (all $p < 0.001$ after correction). The bounded slowdown falls by 84.6% relative to FCFS. Against the two SJF variants, the mean waiting time is 14–16% lower but the difference is not significant, while the 99th-percentile waiting time is 49.6% and 62.3% lower ($p < 0.001$): SJF obtains its mean by starving long tasks, whose maximum waits reach up to 37 s.

Table 4. Mean over ten windows per trace at $\rho = 0.8$ (times in ms; Reconf.: reconfigurations per task; E: energy per task; FI: Jain's index over category slowdowns; Starved: tasks with $w > 5$ s per window). Best values in bold.

Policy	Mean w	P99 w	Max w	BSLD	Premium w	Reconf. (%)	E (mJ)	FI	Starved
SDSC-SP2									
FCFS	853	2,009	2,116	559.7	818	59.9	139.4	0.760	**0.0**
SJF-E	282	5,413	6,743	73.2	353	59.8	140.7	0.803	61.7
SJF-P	273	6,439	16,272	61.0	287	61.0	140.8	**0.809**	56.1
RA-FCFS	256	**1,074**	**1,270**	152.4	235	**27.1**	**135.6**	0.766	**0.0**
DRR	556	2,824	2,980	248.7	100	60.7	139.7	0.609	**0.0**
MLFQ-A	691	2,164	2,385	403.3	562	59.3	139.7	0.748	**0.0**
RL-A-DRR	**127**	1,342	3,363	**41.5**	**47**	37.2	136.5	0.739	2.1
CTC-SP2									
FCFS	963	2,869	3,091	421.5	1,048	64.0	216.6	0.558	**0.0**
SJF-E	481	3,832	4,352	92.7	601	65.5	216.8	0.559	11.7
SJF-P	399	8,035	17,876	**54.3**	587	66.0	216.8	0.584	97.4
RA-FCFS	547	**1,943**	**2,179**	235.4	599	**28.8**	**212.0**	0.549	**0.0**
DRR	649	3,976	4,517	146.4	256	65.8	216.7	0.517	32.4
MLFQ-A	808	2,892	3,288	283.3	769	64.1	216.7	0.582	**0.0**
RL-A-DRR	**325**	2,552	5,852	63.5	**209**	38.0	212.8	**0.585**	6.1
HPC2N									
FCFS	7,279	18,090	18,406	2,886.0	8,255	27.0	359.7	0.588	1,685.6
SJF-E	**2,437**	27,080	36,890	**467.9**	1,883	28.8	360.0	0.525	**328.4**
SJF-P	4,860	26,449	37,009	1,329.0	5,389	32.9	360.7	0.559	634.8
RA-FCFS	6,873	**17,518**	**18,162**	2,606.1	7,832	**13.6**	**357.4**	0.589	1,515.5
DRR	4,379	24,230	24,914	1,056.8	826	33.2	360.1	0.493	904.5
MLFQ-A	7,105	18,038	18,454	2,782.0	7,690	26.5	359.4	**0.599**	1,516.4
RL-A-DRR	5,052	30,241	35,658	2,060.9	**690**	19.3	358.0	0.536	800.5
ANL-Intrepid									
FCFS	1,594	4,017	4,171	481.3	1,683	59.6	225.1	0.556	193.1
SJF-E	481	8,623	11,469	57.7	587	61.7	226.1	0.574	183.3
SJF-P	**431**	9,331	18,461	**35.4**	403	61.4	225.5	0.615	122.7
RA-FCFS	1,072	**3,047**	**3,381**	320.7	1,108	**23.1**	**221.5**	0.553	**46.5**
DRR	1,033	5,317	5,689	168.6	716	62.9	225.2	0.567	265.0
MLFQ-A	1,377	3,962	4,339	366.3	1,370	59.1	225.1	0.549	160.0
RL-A-DRR	519	3,757	8,150	65.0	**329**	31.0	222.2	**0.629**	75.2

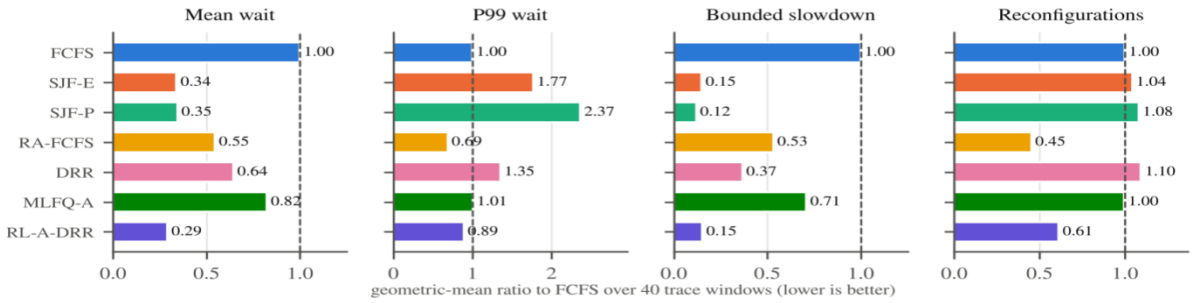


Fig. 3. Geometric mean over the 40 windows of each metric relative to FCFS at $\rho = 0.8$.

Table 5. Paired comparison of RL-A-DRR with each baseline over 40 windows: geometric-mean change and Holm-corrected Wilcoxon p-value (*: $p < 0.05$).

	Mean wait		**P99 wait**		**BSLD**		**Reconfigurations**	
Baseline	change	p	change	p	change	p	change	p
FCFS	−70.8%*	<0.001	−10.9%	0.500	−84.6%*	<0.001	−38.7%*	<0.001
SJF-E	−14.3%	1.000	−49.6%*	<0.001	+2.8%	1.000	−41.2%*	<0.001
SJF-P	−15.5%	1.000	−62.3%*	<0.001	+27.9%	0.095	−43.3%*	<0.001
RA-FCFS	−46.5%*	<0.001	+29.5%*	<0.001	−71.2%*	<0.001	+35.5%*	<0.001
DRR	−54.7%*	<0.001	−34.2%*	<0.001	−58.2%*	<0.001	−44.0%*	<0.001
MLFQ-A	−64.6%*	<0.001	−11.6%	0.500	−78.3%*	<0.001	−38.5%*	<0.001

Reconfiguration affinity removes 38.7% of the reconfigurations relative to FCFS and 44.0% relative to DRR, and with it 38.7% of the reconfiguration energy. Because the execution energy is identical for all work-conserving policies, the energy per task drops by only 1.6% (2.0% for RA-FCFS), and the throughput, which is set by the arrival process, differs by at most 2%. RA-FCFS reuses kernels even more aggressively (RL-A-DRR performs 35.5% more reconfigurations), and it has the shortest tails: its 99th-percentile and maximum waiting times are the lowest on every trace. RL-A-DRR's 99th-percentile wait is 10.9% below FCFS (not significant), but its maximum wait is on average 1.75 times that of FCFS, because the guard intervenes only when a task's wait exceeds κ_g times the current mean. The premium tier is protected best by RL-A-DRR: its mean waiting time is 87.5% below FCFS, against 81.9% for DRR.

The traces stress the policies differently. On SDSC-SP2 and CTC-SP2, RL-A-DRR improves on FCFS, DRR and MLFQ-A in mean waiting time, 99th-percentile waiting time and reconfigurations, and starves 2–6 tasks per window, against 12–97 for the SJF variants. ANL-Intrepid is heavily congested (FCFS waits 64 times $\bar{e}$ on average): RL-A-DRR starves 75 tasks per window against 193 for FCFS and 123–183 for SJF, but 47 for RA-FCFS. HPC2N is dominated by single-user bursts; in one window, one user submits 60% of the tasks and 1,157 tasks arrive within 2 s. There, SJF-E is clearly best on mean waiting time and starvation; RL-A-DRR still reduces the mean wait by 31% relative to FCFS (7.3 s) with 28% fewer reconfigurations and keeps premium tenants at 0.69 s against 8.3 s, but its tail is longer than that of FCFS. A per-tenant rather than per-class hierarchy would isolate such bursts, as we discuss in Section 7.

Load sensitivity

Fig. 4 varies the offered load from 0.6 to 0.95. RL-A-DRR has the lowest mean waiting time at every load on SDSC-SP2 and CTC-SP2 and stays within 13–30% of SJF-P on ANL-Intrepid, while, except on the bursty HPC2N trace, its 99th-percentile waiting time stays close to or below that of FCFS and MLFQ-A instead of growing like that of the SJF variants. On SDSC-SP2 at $\rho = 0.95$, it starves 7 tasks per window against 96 for FCFS and 196–257 for SJF. The ordering of the policies is stable across loads, so the conclusions of Section 6.2 do not depend on the operating point.

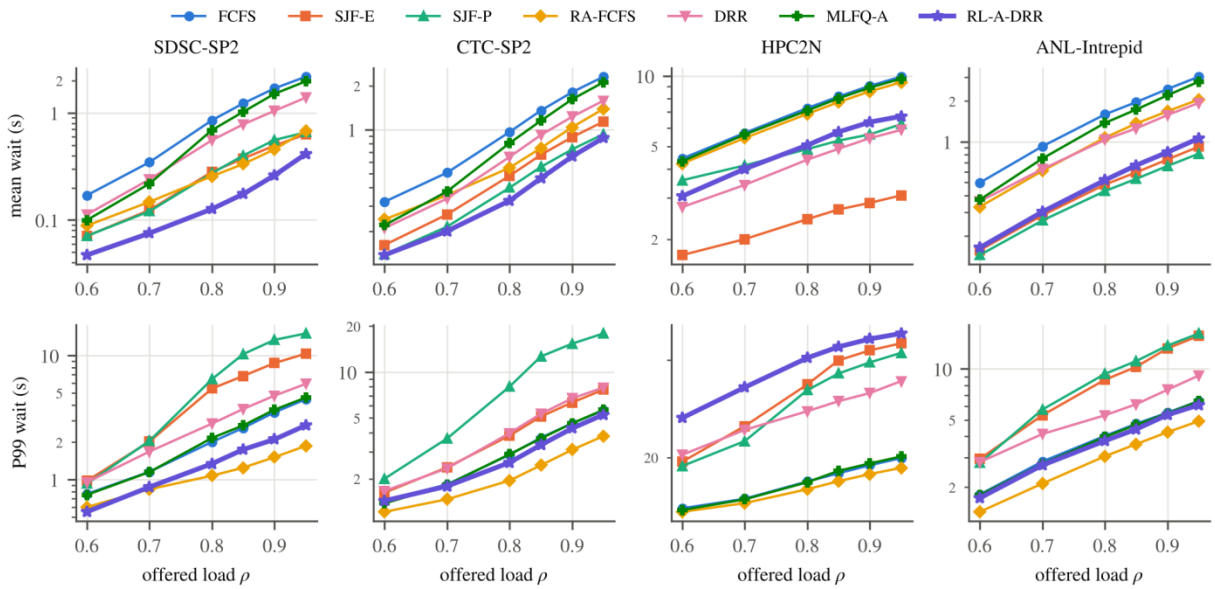


Fig. 4. Mean (top) and 99th-percentile (bottom) waiting time versus offered load (log scale; ten windows per point).

Ablation and robustness to prediction bias

Table 6 removes one mechanism at a time. The look-ahead window with affinity borrowing is the largest contributor: without it, the mean waiting time nearly doubles and reconfigurations increase by 59%. Replacing the residual prediction by the request raises the mean wait by 35% and the bounded slowdown 2.3-fold, because the inflated requests misclassify short tasks as long and blunt the admission order; it does reduce reconfigurations, since inflated costs leave abundant credit for reuse. Online refinement is worth 1–4%. Without the guard, the maximum waiting time grows by 55%; without aging, the 99th-percentile wait and reconfigurations increase by 8% and 11%. Aging and the guard deviate from strict shares, which is visible in the larger relative fairness bound (RFB, the largest weighted service gap between jointly backlogged classes, in multiples of $\bar{e}$). Error compensation barely changes the efficiency metrics, but without it the RFB is 95% larger and the premium tier waits 51% longer, even without injected bias.

Table 6. Ablation over 40 windows: geometric mean of each metric relative to the full RL-A-DRR (< 1 is better, except that RFB measures share fidelity).

Variant	Mean w	P99 w	Max w	BSLD	Reconf.	Premium w	RFB
RL-A-DRR (full)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
w/o residual learning (cost = request)	1.35	1.09	0.85	2.27	0.69	2.52	3.53
w/o online residual feedback	1.01	1.03	1.01	0.99	1.04	0.92	0.79
w/o error-compensated deficit	1.04	0.99	0.96	1.18	0.87	1.51	1.95
w/o affinity window ($W = 1$, no borrowing)	1.94	1.36	0.73	2.13	1.59	1.63	0.98
w/o aging credit boost	1.05	1.08	1.04	0.97	1.11	0.77	0.29
w/o starvation guard	0.97	0.99	1.55	0.95	1.00	0.96	0.99
w/o aging and guard	1.01	1.07	1.65	0.88	1.12	0.70	0.23
classic DRR (reference)	2.21	1.52	0.79	2.39	1.79	1.44	0.58

Fig. 5 tests Proposition 1 directly by multiplying the admission cost of all low-tier tasks by $b \in \{1, 0.5, 0.25, 0.1\}$, which models a mis-calibrated predictor or tenants that under-report. With error compensation, the premium tier's mean wait at $b = 0.1$ changes by -4% to $+38\%$ relative to $b = 1$. Without compensation, it grows 1.8–4.6 times, and with classic DRR, whose costs are the requests themselves, 3.7–19.7 times. The true-up thus converts an assumption about prediction accuracy into a measured property.

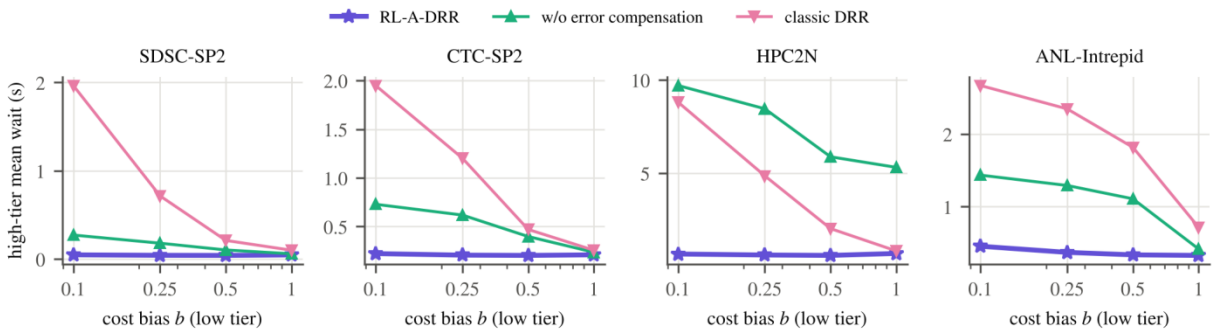


Fig. 5. Premium-tier mean waiting time when low-tier costs are under-estimated by the factor b.

Parameter sensitivity

Table 7 varies one parameter at a time around the defaults over the 40 evaluation windows. The window is the most influential parameter: $W = 1$ raises the mean wait by 78%, and the benefit saturates at $W = 16$–32, where larger windows trade the 99th percentile for the mean. Borrowing mainly reduces reconfigurations: $B = 16$ performs 17% fewer than the default and 34% fewer than $B = 0$, with little change in the mean. The guard factor trades the mean against the maximum wait ($\kappa_g = 4$ lowers the maximum by 23% for an 8% higher mean). The aging factor changes the metrics by at most 5%, the boost factor by at most 5% for $\alpha \leq 1.5$ (a large boost, $\alpha = 2$, raises the mean by 10%), and strongly differentiated weights $(1,0.6,0.8,0.4)$ cost 6% in mean waiting time relative to the default weights.

Table 7. Sensitivity: geometric mean over 40 windows relative to the default (bold).

Parameter	**Value**	**Mean** w	**P99** w	**Max** w	**Reconf.**
Look-ahead window W	1	1.78	1.27	0.69	1.44
	4	1.33	1.08	0.91	1.19
	8	1.16	1.03	1.00	1.08
	16	1.00	1.00	1.00	1.00
	32	0.91	1.14	1.00	0.96
Affinity borrowing B (quanta)	0	1.11	1.17	1.10	1.25
	1	1.05	1.08	1.06	1.14
	2	1.02	1.05	1.06	1.09
	4	1.00	1.00	1.00	1.00
	8	0.99	0.98	0.93	0.91
	16	1.01	0.96	0.90	0.83
Aging factor κ_a	1	1.04	0.97	0.93	0.92
	2	1.00	1.00	1.00	1.00
	4	1.01	1.03	1.01	1.03
	8	1.01	1.05	1.03	1.05
Guard factor κ_g	4	1.08	1.02	0.77	1.03
	6	1.02	1.01	0.92	1.01
	8	1.00	1.00	1.00	1.00
	12	0.99	1.01	1.12	1.01
	16	0.99	1.01	1.21	1.00
Boost factor α	0.5	1.02	1.05	1.05	1.09
	1.0	1.01	1.02	1.01	1.03
	1.25	1.00	1.00	1.00	1.00
	1.5	1.02	1.00	1.00	0.97
	2.0	1.10	0.98	0.96	0.90
Class weights	equal	1.00	0.97	0.96	0.91
	mild	1.00	1.00	1.00	1.00
	strong	1.06	1.09	1.07	1.13

Overhead and scalability

At the default configuration, an RL-A-DRR decision takes 0.80 μs on average and 2.9 μs at the 99th percentile, of which 0.12 μs is aging and guard bookkeeping and the rest is the windowed scan. Inference at arrival costs 0.35 μs, the online gradient step 0.74 μs and the true-up 25 ns. These costs are more than three orders of magnitude below the 4 ms reconfiguration and well below the sub-millisecond budget usually assumed for software schedulers. Table 8 confirms that the decision latency does not grow with the window size N (0.68–0.76 μs from 1,000 to 20,000 tasks), as the windowed admission implies, and that it grows roughly linearly with the number of regions, because residency is evaluated per region. The baselines decide in 35–170 ns; a hardware implementation such as that of Jiang et al. [23] could narrow the gap further.

Table 8. Mean decision latency (ns) versus window size N ($S = 4$) and number of regions S ($N = 5{,}000$); five windows per trace.

Configuration	FCFS	SJF-P	DRR	MLFQ-A	RA-FCFS	RL-A-DRR
N = 1,000	42	77	103	61	75	682
N = 2,500	37	84	72	60	75	702
N = 5,000	37	85	68	62	71	738
N = 10,000	35	86	95	60	70	733
N = 20,000	35	91	73	60	73	759
S = 2	35	76	67	55	61	710
S = 4	41	86	71	62	83	756
S = 8	42	103	86	72	104	1229
S = 16	63	157	104	93	170	1623

More regions expose the ICAP bottleneck. At the same load, the mean waiting time of FCFS falls from 1.96 s with 4 regions to 1.60 s with 8 but rises to 2.48 s with 16, because reconfigurations queue behind the single port; RL-A-DRR improves monotonically (0.82, 0.44 and 0.34 s), as does RA-FCFS (1.43, 0.77 and 0.61 s).

Heterogeneous Big.Little regions

Table 9 repeats the comparison on the Big.Little platform. RL-A-DRR reduces the mean waiting time to 0.22 of FCFS, the best of all policies, and the 99th-percentile wait to 0.67, while RA-FCFS again has the shortest tail and the fewest reconfigurations. The completion-time-aware region choice makes no measurable difference relative to residency-first selection with least-recently-used eviction (last row), which suggests that with a single ICAP the benefit of reuse outweighs that of matching tasks to fast regions; matching becomes relevant only when reconfiguration is cheap.

Table 9. Big.Little platform at $\rho = 0.8$: mean waiting time (ms) per trace and geometric-mean ratios to FCFS over 40 windows.

	Mean w (ms)				Ratio to FCFS			
Policy	SDSC	CTC	HPC2N	ANL	Mean w	P99 w	Recon f.	E
FCFS	2,291	1,655	7,030	2,346	1.00	1.00	1.00	1.00
SJF-E	807	970	2,394	774	0.38	1.96	1.03	1.01
SJF-P	917	790	4,731	705	0.39	2.83	1.08	1.01
RA-FCFS	304	602	6,374	1,125	0.36	0.47	0.43	0.95
DRR	1,669	1,257	4,399	1,586	0.70	1.46	1.09	1.00
MLFQ-A	2,118	1,522	6,922	2,085	0.90	1.00	1.00	1.00
RL-A-DRR	204	386	4,736	569	0.22	0.67	0.56	0.96
RL-A-DRR w/o het.-aware choice	189	387	4,844	568	0.22	0.67	0.56	0.96

Threats to validity and limitations

Workload mapping. The traces record HPC jobs, not FPGA tasks. We keep what transfers, namely the joint distribution of requests and actual times, user histories, arrival burstiness and reuse locality, and we synthesise what does not, namely kernels, tiers and time scale, with fixed, documented hashes. The mapping uses one region per task and ignores requested parallelism except for kernel identity. Results for accelerators whose execution times are near-deterministic would favour all prediction-based policies equally.

Simulation. Reconfiguration time follows the bitstream size and the ICAP bandwidth, without the latency variations of real ports, and the power model is parameterised rather than measured. Hardware validation on a DFX shell such as Coyote v2 [3] is the natural next step; the scheduler's state fits in a few kilobytes and its inference in about 160 multiply–accumulates, which is compatible with a soft processor or a hardware scheduler [23].

Tuning. Parameters were chosen on a separate trace, but the four evaluation traces share the HPC origin; the sensitivity study shows that the conclusions hold over wide parameter ranges.

Limitations. Fairness is enforced between four classes, not between tenants, so a burst of one tenant competes within its class (HPC2N); a two-level hierarchy with per-tenant deficits inside each class would address this at $O(\log U)$ extra cost. RA-FCFS remains better on tail latency and reconfiguration count, and a combined objective that weighs reuse against service order explicitly is an open design point. Dependent tasks (DAGs) can be supported by admitting only ready tasks into the class queues and by adding the critical-path length to the classification, without changing the deficit mechanism.

Conclusion

RL-A-DRR couples a lightweight residual predictor of execution time with an approximate, reconfiguration-affine and error-compensated deficit round robin for multi-tenant partially reconfigurable FPGAs. On 200,000 tasks from four production traces it reduced the mean waiting time by 70.8% relative to FCFS and by 46.5–64.6% relative to reconfiguration-aware FCFS, classic DRR and MLFQ with aging, matched SJF on mean waiting time while lowering its 99th-percentile waiting time by 50–62%, reduced reconfigurations by 39% and protected premium tenants best. Error compensation made the premium tier's service insensitive to a tenfold cost under-estimation, which is the property that matters when predictions are imperfect, as they are on real workloads. Decisions take under a microsecond. Future work will add per-tenant hierarchy, validate the scheduler on a DFX shell, and use HLS-based latency models to initialise the request of new kernels.

CRediT authorship contribution statement

Abirami B: Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Visualization, Writing – original draft. **V. Vasudevan:** Conceptualization, Validation, Resources, Supervision, Project administration, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data and code availability

The workload traces are publicly available from the Parallel Workloads Archive [39]. The C source code of the simulator, the predictors and the schedulers, the SHA-256 checksums of the traces, and the scripts that regenerate all results are provided as supplementary material and will be made available in a public repository upon publication.

References

1. AMD, *Vivado Design Suite User Guide: Dynamic Function eXchange (UG909)*. Advanced Micro Devices, Inc., 2023.

2. A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 13–24, 2014.

3. B. Ramhorst, D. Korolija, M. J. Heer, J. Dann, L. Liu, and G. Alonso, “Coyote v2: Raising the level of abstraction for data center FPGAs.” arXiv preprint arXiv:2504.21538, 2025.

4. A. Koshiba, C. Mainas, and P. Bhatotia, “Funky: Cloud-native FPGA virtualization and orchestration,” in *Proceedings of the ACM Symposium on Cloud Computing (SoCC ’25)*, 2025.

5. J. Gu, H. Wang, X. Guo, M. Schulz, and M. Gerndt, “VersaSlot: Efficient fine-grained FPGA sharing with Big.Little slots and live migration in FPGA cluster,” in *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025.

6. E. Karabulut, A. A. Malik, A. Awad, and A. Aysu, “THEMIS: Time, heterogeneity, and energy minded scheduling for fair multi-tenant use in FPGAs,” *IEEE Transactions on Computers*, vol. 74, no. 7, pp. 2515–2528, 2025.

7. A. A. Malik, E. Karabulut, and A. Aysu, “EPOCH: Enabling preemption operation for context saving in heterogeneous FPGA systems.” arXiv preprint arXiv:2501.16205, 2025.

8. W. Shen, M. Han, J. Liu, R. Chen, and H. Chen, “XSched: Preemptive scheduling for diverse XPUs,” in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, USENIX Association, 2025.

9. K. Papadimitriou, A. Dollas, and S. Hauck, “Performance of partial reconfiguration in FPGA systems: A survey and a cost model,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, no. 4, pp. 36:1–36:24, 2011.

10. D. Tsafrir, Y. Etsion, and D. G. Feitelson, “Backfilling using system-generated predictions rather than user runtime estimates,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

11. R. Quispe-Vargas, D. M. Yana-Yucra, R. A. Vilca-Solorzano, V. Ibañez-Quispe, and F. Torres-Cruz, “Machine learning-based runtime prediction and energy optimization for HPC job scheduling using the NREL Eagle supercomputer dataset,” *Informatica*, vol. 50, no. 15, pp. 44–60, 2026.

12. M. Shreedhar and G. Varghese, “Efficient fair queuing using deficit round-robin,” *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, 1996.

13. M. Mandava, P. Reckamp, and D. Chen, “Nimblock: Scheduling for fine-grained FPGA sharing through virtualization,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA ’23)*, ACM, 2023.

14. A. Dhar, E. Richter, M. Yu, W. Zuo, X. Wang, N. S. Kim, and D. Chen, “DML: Dynamic partial reconfiguration with scalable task scheduling for multi-applications on FPGAs,” *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2577–2591, 2022.

15. M. Mitzenmacher and S. Vassilvitskii, “Algorithms with predictions,” *Communications of the ACM*, vol. 65, no. 7, pp. 33–35, 2022.

16. Y. Fu, S. Zhu, R. Su, A. Qiao, I. Stoica, and H. Zhang, “Efficient LLM scheduling by learning to rank,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

17. Z. Luo, J. Liu, M. Lee, and N. B. Shroff, “Prediction-assisted online distributed deep learning workload scheduling in GPU clusters,” in *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1–10, 2025.

18. A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach, “Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 107–127, 2018.

19. Y. Zha and J. Li, “Virtualizing FPGAs in the cloud,” in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’20)*, pp. 845–858, 2020.

20. D. Korolija, T. Roscoe, and G. Alonso, “Do OS abstractions make sense on FPGAs?,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 991–1010, 2020.

21. C. Heinz, J. Hofmann, J. Korinth, L. Sommer, L. Weber, and A. Koch, “The TaPaSCo open-source toolflow for the automated composition of task-based parallel reconfigurable computing systems,” *Journal of Signal Processing Systems*, vol. 93, pp. 545–563, 2021.

22. A. Rodríguez, A. Navarro, K. Nikov, J. Nunez-Yanez, R. Gran, D. S. Gracia, and R. Asenjo, “Lightweight asynchronous scheduling in heterogeneous reconfigurable systems,” *Journal of Systems Architecture*, vol. 124, p. 102398, 2022.

23. L. Jiang, Z. Ou, and D. Göhringer, “Hardware-level adaptive scheduling for reconfigurable accelerators on virtualized FPGAs,” in *28th Euromicro Conference on Digital System Design (DSD)*, pp. 566–573, 2025.

24. A. Zou, Y. Xu, Y. Ni, J. Chen, Y. Ma, J. Li, C. Gill, X. Zhang, and Y. Jin, “A survey of real-time scheduling on accelerator-based heterogeneous architecture for time critical applications.” arXiv preprint arXiv:2505.11970, 2025.

25. A. Patras, S. Lalis, C. D. Antonopoulos, and N. Bellas, “DPUConfig: Optimizing ML inference in FPGAs using reinforcement learning.” arXiv preprint arXiv:2602.12847, 2026.

26. E. Russo, F. G. Blanco, M. Palesi, G. Ascia, D. Patti, and V. Catania, “Towards fair and firm real-time scheduling in DNN multi-tenant multi-accelerator systems via reinforcement learning,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2024.

27. E. Sharafzadeh, R. Matson, J. Tourrilhes, P. Sharma, and S. Ghorbani, “Self-clocked round-robin packet scheduling,” in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 1437–1465, 2025.

28. Y. Sheng, S. Cao, D. Li, B. Zhu, Z. Li, D. Zhuo, J. E. Gonzalez, and I. Stoica, “Fairness in serving large language models,” in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, 2024.

29. R. Jain, D.-M. Chiu, and W. R. Hawe, “A quantitative measure of fairness and discrimination for resource allocation in shared computer systems,” Tech. Rep. DEC-TR-301, Digital Equipment Corporation, 1984.

30. Z. Benomar and V. Perchet, “Non-clairvoyant scheduling with partial predictions,” in *Proceedings of the 41st International Conference on Machine Learning (ICML)*, vol. 235 of *PMLR*, pp. 3506–3538, 2024.

31. T. Zhao, C. Xia, X. Chang, C. Li, W. Li, and A. Zomaya, “Competitive fair scheduling with predictions,” in *International Conference on Learning Representations (ICLR)*, 2025.

32. Z. Ye, W. Gao, Q. Hu, P. Sun, X. Wang, Y. Luo, T. Zhang, and Y. Wen, “Deep learning workload scheduling in GPU datacenters: A survey,” *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024.

33. Green, K., & Vrba, R. (2026). Secure bitstream management and authentication framework for dynamic FPGA reconfiguration. *Journal of Reconfigurable Hardware Architectures and Embedded Systems*, 3(2), 36–45.

34. E. Kolker-Hicks, D. Zhang, and D. Dai, “A reinforcement learning based backfilling strategy for HPC batch jobs,” in *Proceedings of the SC ’23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (PMBS)*, 2023.

35. K. Menear, K. Konate, K. Potter, and D. Duplyakin, “Tandem predictions for HPC jobs,” in *Practice and Experience in Advanced Research Computing (PEARC ’24)*, pp. 1–9, 2024.

36. M. Gao, J. Zhao, Z. Lin, and M. Guo, “Hierarchical source-to-post-route QoR prediction in high-level synthesis with GNNs.” arXiv preprint arXiv:2401.08696, 2024.

37. A. W. Mu’alem and D. G. Feitelson, “Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

38. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, 2015.

39. D. G. Feitelson, D. Tsafrir, and D. Krakov, “Experience with using the Parallel Workloads Archive,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014.

40. F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

41. S. Holm, “A simple sequentially rejective multiple test procedure,” *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

42. Green, K., & Vrba, R. (2026). Secure bitstream management and authentication framework for dynamic FPGA reconfiguration. *Journal of Reconfigurable Hardware Architectures and Embedded Systems*, 3(2), 36–45.

43. Madhanraj. (2026). Doppler-Aware Subspace Tracking With Pilot Reliability Weighting For Low-Complexity Massive-Mimo Channel Estimation. *VSF Journal of Electronics and Communication Engineering*, *1*(1), 38–44.